SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Diplomski rad

Ocjena učinkovitosti operatora križanja u genetskom programiranju

Maja Kontrec

Zagreb, 2014.

Sadržaj

Uvod

Genetsko programiranje predstavlja optimizacijsku tehniku iz skupine evolucijskih algoritama. Ono omogućava rješavanje teških optimizacijskih problema za koje, ili ne postoji egzaktna matematička metoda rješavanja, ili su nerješivi u nekom konačnom vremenu.

Temeljeći se na imitaciji prirodnog evolucijskog procesa, ovaj algoritam djeluje uporabom genetskih operatora križanja, mutacije i selekcije. Genetski operatori u svakoj iteraciji modificiraju trenutnu populaciju potencijalnih rješenja - jedinki. Pri tome, operatori mutacije i križanja služe za pretraživanje samog prostora rješenja, dok selekcija služi kako bi bolje jedinke imale veću vjerojatnost preživljavanja nad onim lošijim. Ovime, genetsko programiranje obuhvaća veći dio prostora pretrage nego srodni algoritmi, koji unutar prostora svih mogućih rješenja pretražuju samo ona koja trenutno susjedna. Još jedna velika prednost genetskog programiranja nad drugim tehnikama optimizacije je ta da je ono neovisno o domeni, odnosno, može se koristiti za rješavanje širokog skupa različitih problema.

Unatoč navedenim prednostima genetskog programiranja, kao i genetskih algoritama općenito, danas još uvijek nije jednoznačan postupak odabira specifičnih podvrsta genetskih operatora, kao ni samog prikaza potencijalnih rješenja. Postupak postavljanja algoritma oslanja se na iskustvo samog postavljača te se temelji na eksperimentiranju. Eksperimentiranjem, potrebno je utvrditi koji su genetski operatori, zajedno s njihovim specifičnim parametrima, dobri za rješavanje problema.

U ovom radu, dan je pregled i opis postojećih genetskih operatora s naglaskom na operatore križanja. Ispitana je učinkovitost algoritma genetskog programiranja s obzirom na odabir operatora križanja za tri najčešće vrste problema koji se rješavaju genetskim programiranjem - simboličku regresiju, pronalazak logičkih funkcija i programe.

Genetsko programiranje

Genetsko programiranje podvrsta je genetskog algoritma. Najveća razlika između genetskog programiranja i genetskog algoritma je ta da je kod genetskog algoritma rješenje, odnosno jedinka, nepromjenjive duljine tijekom čitavog procesa evolucije.

U genetskom algoritmu, jedinka je predstavljena nizom informacija, primjerice nizom znamenki ili znakova, dok je kod genetskog programiranja jedinka predstavljena nekim slobodnijim oblikom, najčešće stablastom strukturom. Stablasta struktura pritom može predstavljati izvršivi program ili pak neki logički ili matematički izraz. Osim najčešće stablaste strukture, jedinka genetskog programiranja također može biti predstavljena i linearno, nizom informacija promjenjive duljine (primjerice, u slučaju da jedinka predstavlja program, može biti predstavljena nizom izvršivih instrukcija).

2.1 Tijek algoritma

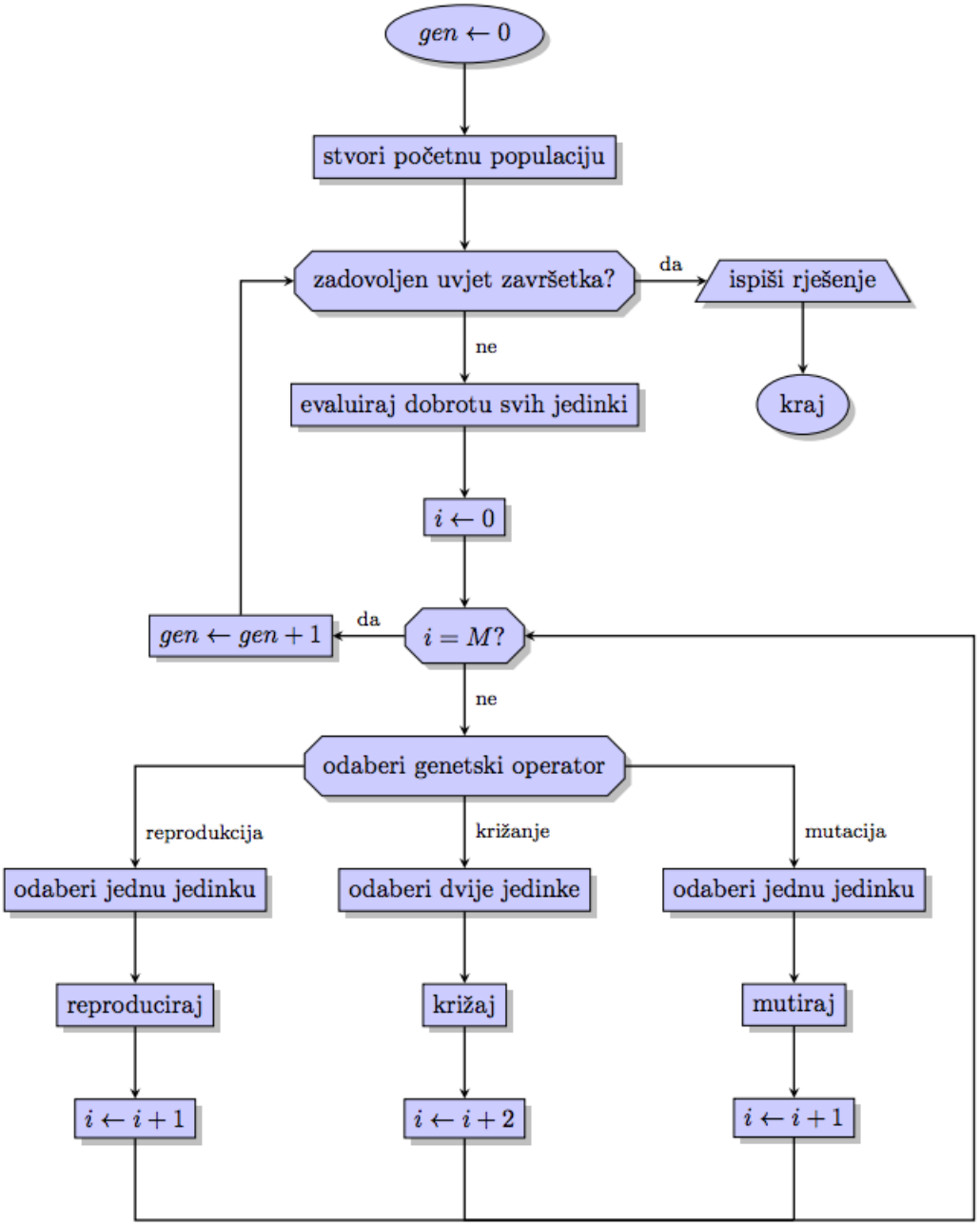
Sam algoritam genetskog programiranja oponaša prirodnu evoluciju. Nakon stvaranja, skup jedinki koji čini populaciju algoritma, prolazi kroz proces evolucije pronalazeći rješenje danog problema. Tijek algoritma prikazan je na slici 2.1.

Algoritam započinje inicijalizacijom početne populacije rješenja, odnosno jedinki. Nakon stvaranje početne populacije, potrebno je evaluirati svaku zasebnu jedinku. Evaluacija se pritom odnosi na izračun dobrote (*eng. fitness*) jedinke; svojstva koje govori o tome koliko je ona kvalitetno rješenje. Pomoću dobrote, jedinke su međusobno usporedive.

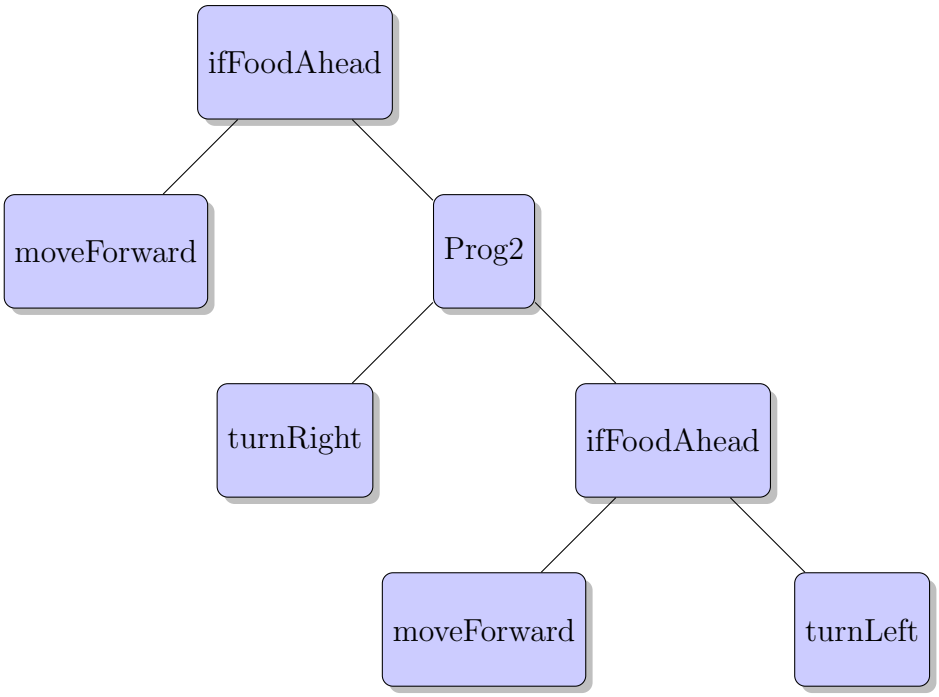
Inicijalizacijom populacije započinje prva generacija algoritma. Na početku svake generacije provjerava se da li je zadovoljen kriterij zaustavljanja algoritma. Ukoliko je, algoritam završava sa svojim izvođenjem, a u suprotnom, sljedeći korak algoritma je selekcija jedinki. Selekcijom se odabiru jedinke koje će preživjeti i sudjelovati u križanju u ulozi roditelja, te jedinka koja će biti zamijenjena njihovim potomkom.

Nakon dobivanja nove jedinke križanjem, ona se mutira s određenom vjerojatnosti mutacije te se izračunava njena nova dobrota.

Ova tri operatora; selekcija, mutacija i križanje, predstavljaju imitaciju evolucije u samom algoritmu. Isto kao u prirodi, u sljedeću generaciju ulaze samo najjače jedinke čiji se genetski materijal kombinira u nešto novo, pokušavajući naći kombinaciju materijala koja će donijeti poboljšanje.



Slika 2.1: Tijek izvršavanja algoritma genetskog programiranja (slika preuzeta iz [1])



Slika 2.2: Primjer jedinke programa umjetnog mrava

2.2 Stablasta jedinka

Najčešći oblik jedinke korišten u genetskom programiranju je stablo. Razlog zašto je tome tako je činjenica da se stablo može interpretirati na više različitih način. Stablo je pogodno za prikaz programskih jedinki, kao i prikaz jedinki koji predstavljaju neki matematički ili logički izraz.

2.2.1 Stablo kao program

Pomoću stabla lako je modelirati neki izvršivi program. Jedan od poznatijih problema rješavanih genetskim programiranjem je problem umjetnog mrava (*eng. Artificial ant problem* [2]). U ovom slučaju, jedinka predstavlja ponašanje mrava unutar neke okoline. Cilj algoritma je evoluirati ponašanje koje će efikasno pronalaziti hranu unutar te okoline.

Nezavršni znakovi unutar stabla pritom donose odluke o ponašanju, dok završni znakovi predstavljaju samo ponašanje pojedinog mrava. Na slici 4.3 prikazana je jedna takva jedinka.

Nezavršni znakovi koji grade jednu ovakvu jedinku su:

1. **If food ahead** - provjerava da li se nalazi hrana ispred mrava - ako da izvršava lijevu granu, ako ne, izvršava desnu granu.
2. **Prog2** - slijedno izvršava lijevu, te zatim desnu granu.
3. **Prog3** - slijedno izvršava (s lijeva na desno) svaku od 3 pridružene grane.

Završni znakovi koji grade ovu jedinku su:

1. **move forward** - pomiče mrava za jedno mjesto unaprijed.
2. **turn right** - okreće mrava u desno.
3. **turn left** - okreće mrava u lijevo.

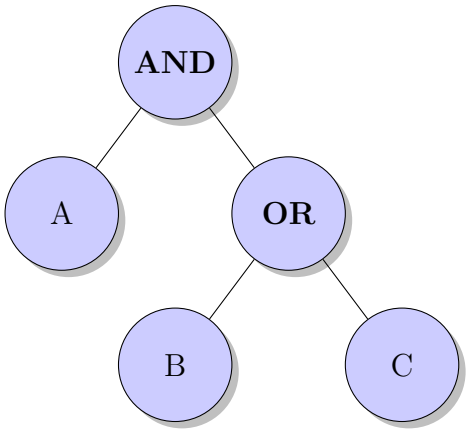
Ovakva stabla vrlo je jednostavno za interpretirati - kreće se od vršnog čvora, te se prema pravilima čvorova prolazi kroz stablo i izvršavaju dane instrukcije. Pseudokod koji opisuje stablo prikazano na slici 4.3 dan je ispod (1).

```
if food ahead then
 | move forward;
else
 | turn right;
 | if food ahead then
 |  | move forward;
 | else
 |  | turn left;
 | end
end
```

Algoritam 1: Pseudokod jedinke prikazane na slici 4.3

2.2.2 Stablo kao logički izraz

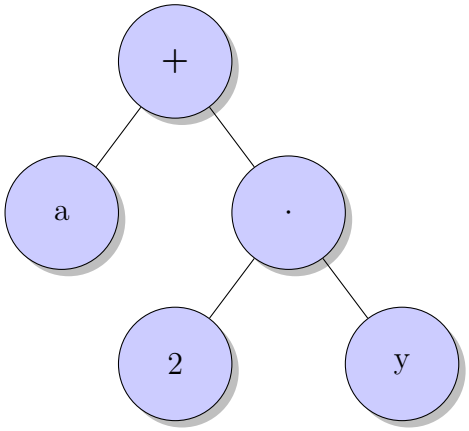
Osim programa, stablo može reprezentirati i neki logički izraz. Pri tome, nezavršni čvorovi predstavljaju logičke funkcije, dok završni čvorovi predstavljaju varijable. Logička funkcija predstavljena takvim stablom jednostavno se može izlučiti *in-order* obilaskom stabla. Jedna ovakva jedinka dana je na slici 2.3.



Slika 2.3: Stablo koje predstavlja logički izraz $A \cdot (B + C)$

2.2.3 Stablo kao matematički izraz

Jednako kao i logičke izraze, stablo može predstavljati neki matematički izraz. Kao i u prethodnom slučaju, matematički izraz se iz stabla može izlučiti *in-order* obilaskom. Na slici 2.4 prikazana je jedna takva jedinka.



Slika 2.4: Stablo koje predstavlja matematički izraz $a + 2 \cdot y$

2.3 Dobrota jedinke

Dobrota (*eng. fitness*) jedinke predstavlja njenu kvalitetu - označava koliko dobro ta jedinka rješava dani problem. Dobrota je najčešće predstavljena realnim brojem, te omogućuje međusobnu usporedivost jedinki. Na osnovi iznosa dobrote pojedine jedinke provodi se selekcija unutar populacije.

Definiranje funkcije dobrote kojom se dobiva sam iznos dobrote jedan je od problema koji se pojavljuje prilikom postavljanja samog algoritma. Funkcija dobrote razlikuje se za svaki pojedini problem.

Lako je uočljivo kako dobrota uvelike definira smjer napretka algoritma, te je stoga pitanje pronalaska adekvatne funkcije dobrote vrlo važno za kvalitetu samog algoritma.

2.4 Operatori selekcije

Selekcija u genetskom programiranju osigurava prijenos kvalitetnog genetskog materijala u buduće generacije birajući jedinke koje će sudjelovati u reprodukciji [3]. Operatori selekcije se prema metodi odabira jedinki dijele na:

- proporcionalne selekcije - odabiru jedinke s vjerojatnošću koja je proporcionalna njihovoj dobroti
- rangirajuće selekcije - odabiru jedinke s vjerojatnošću koja ovisi o rangu dobrote jedinke (u usporedbi s dobrotama ostalih jedinki)

Ovisno o načinu prenošenja genetskog materijala boljih jedinki u iduću iteraciju načini selekcije se dijele na:

- generacijske selekcije - selekcija direktno bira bolje jedinke koje sudjeluju u križanju
- eliminacijske selekcije - selekcija bira lošije jedinke za eliminaciju; bolje jedinke u ovom slučaju preživljavaju selekciju

2.4.1 Generacijska selekcija

Generacijska selekcija direktno odabire dvije jedinke koje će sudjelovati u reprodukciji, kopirajući one bolje jedinke u novu populaciju - nazvanu međupopulacijom. Broj jedinki u međupopulaciji manji je nego onaj u originalnog populaciji. Budući da sve jedinke populacije neće preživjeti selekciju, u međupopulaciju se najčešće stavljaju duplikati preživjelih jedinki. Iz ovoga se vidi očiti problem koji nastaje ovakvom selekcijom - smanjuje se raznovrsnost genetskog materijala koji će sudjelovati u repro-

dukciji. Posljedično, nedostatak genetske raznovrsnosti doprinosi usporavanju procesa evolucije.

Iz opisanog je vidljivo kako generacijska selekcija stvara čvrstu granicu između generacija - svaka jedinka postoji točno jednu generaciju.

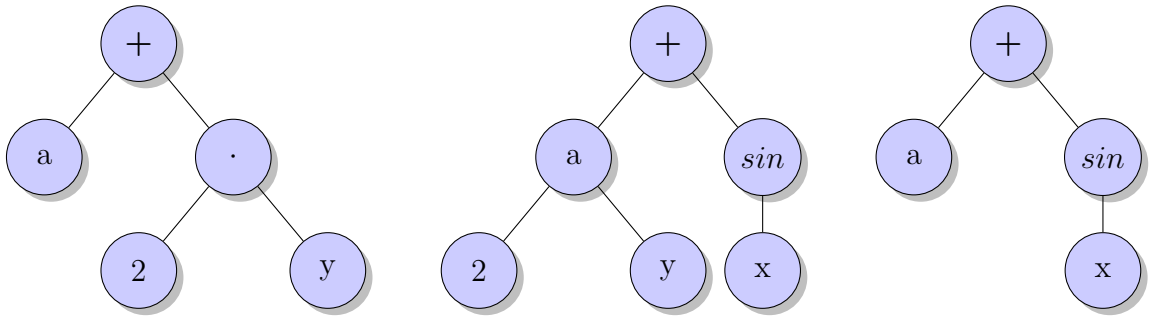
2.4.2 Eliminacijska selekcija

Za razliku od generacijske selekcije, kod koje postoji čvrsta granica između generacija, u eliminacijskoj selekciji dobre jedinke preživljavaju više generacija. Eliminacijska selekcija nadomješta M jedinki novim jedinkama koje su nastale reprodukcijom onih boljih. Pri tome, parametar M se naziva mortalitetom.

Eliminacijska selekcija svojim djelovanjem ne stvara duplikate jedinki čime se otklanja osnovni problem generacijske selekcije.

2.5 Operatori križanja

Operatori križanja obavljaju rekombinaciju genetskog materijala, stvarajući jednu novu jedinku koja nastaje kombinacijom neke druge dvije, roditeljske jedinke. U slučaju genetskog programiranja gdje je jedinka predstavljena u obliku stabla, križanje se svodi na spajanje dvaju podstabla roditelja u novo stablo djeteta (slika 2.5). Zasebni operatori križanja detaljno će biti opisani u idućem poglavlju.



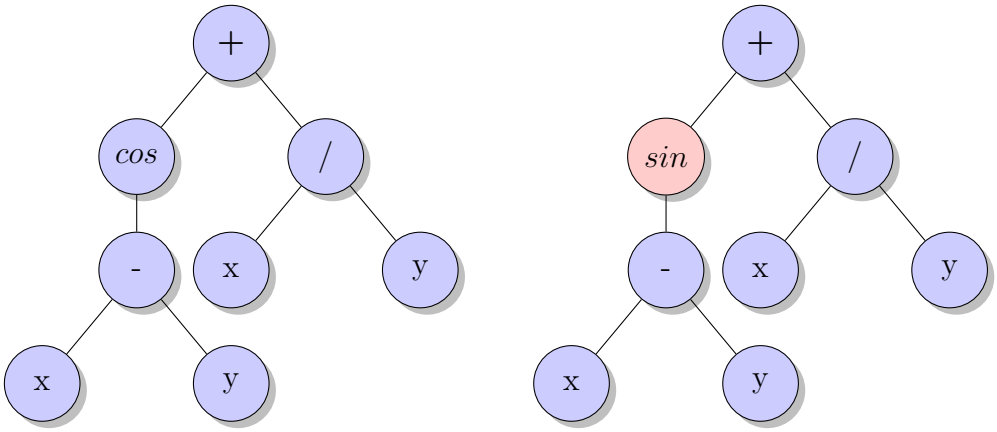
Slika 2.5: Primjer jednostavnog križanja stabala

2.6 Operatori mutacije

Mutacija je operator koji djeluje direktno na pojedinu jedinku. Kao i u prirodi, mijenja jedan dio jedinke. Jedinka mutira s određenom vjerojatnošću, koji je uobičajeno broj reda veličine 0.1. Na slici 2.6 prikazana je jednostavna mutacija koja nasumično mijenja jedan čvor stabla.

Uobičajene metode mutacije uključuju:

- zamjena nasumično odabranog podstabla novim, također nasumično generiranim stablom
- zamjena nasumično odabranog čvora njegovim komplementom
- zamjena slučajno odabranog čvora čvorom istog tipa
- zamjena slučajno odabranog podstabla završnim čvorom



Slika 2.6: Primjer jednostavne mutacije. Desno: jedinka prije mutacije, lijevo: jedinka nakon mutacije

Mutacija je vrlo korisna u slučaju kada jedinka zapne u nekom od lokalnih minimuma. Naime, mutacija može značajno promijeniti jedinku, te ju tako dosta udaljiti od trenutnog prostora pretraživanja.

Operatori križanja

Utjecaj operatora križanja već je neko vrijeme jedna od zanimljivijih tema genetskog programiranja. Često se postavlja pitanje je li uopće algoritam učinkovitiji kada se koriste operatori križanja ili je jednostavnije i bolje jednostavno odbaciti takav način i okrenuti se nekim drugim pristupima koji posjeduju veću razinu determinizma.

Na ovu temu, provedeno je nekoliko istraživanja. John R. Koza u [2] je uključio jednostavnu usporedbu algoritama genetskog programiranja sa i bez operatora križanja (koristeći mutaciju u oba slučaja), te pokazao da postojanje operatora križanja uvelike doprinosi kvaliteti konačnog rješenja. Međutim, istraživanje nije detaljno opisao niti je spomenuo da je koristio neke statističke metodi pri tome. Luke i Spector [4] objavili su usporedbu operatora križanja i mutacije na četiri različita problema. Nisu našli nikakve važnije razlike između ekskluzivnog korištenja mutacije ili križanja. White i Poulding [5] zaključili su da operator križanja nema preveliku prednost pred operatorom mutacije.

U sljedećim poglavljima, detaljno će biti opisani različiti operatori križanja u genetskom programiranju, te će biti opisana do sada provedena istraživanja sa spomenutim operatorima. Ova istraživanja, osim predstavljanja pojedinog operatora, uključuju i njegovu usporedbu s ostalim operatorima križanja. U kasnijim poglavljima biti će predstavljena istraživanja, zajedno s njihovim zaključcima, provedena u sklopu izrade ovoga rada.

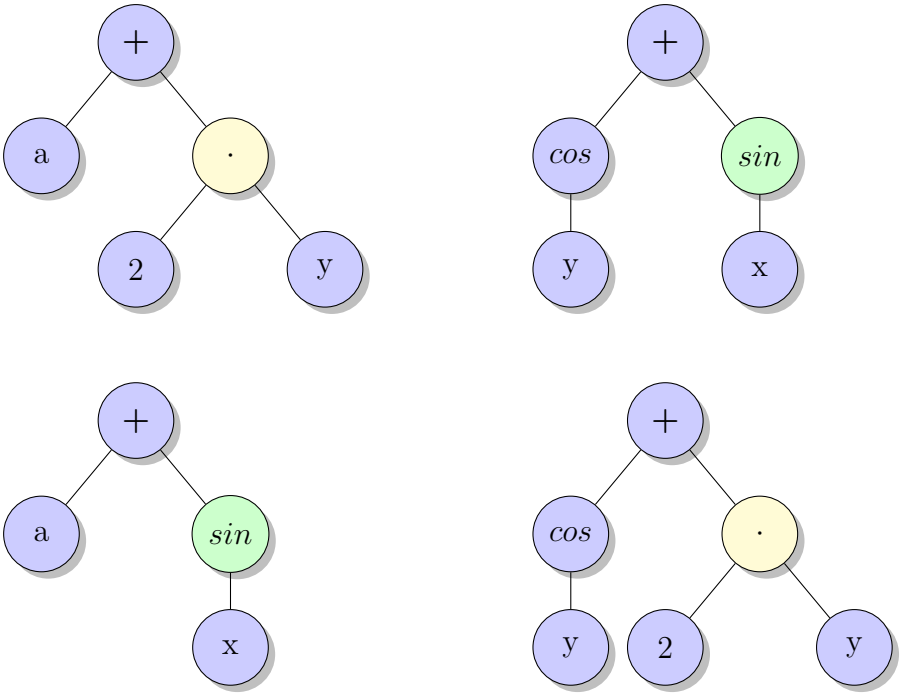
3.1 Jednostavno križanje

Jednostavno križanje (*eng. subtree crossover*) predstavljeno je u [6]. Ovo križanje ujedno je i najjednostavnije i najkorištenije križanje u genetskom programiranju.

Nova jedinka nastaje spajanjem dvaju nasumično odabranih podstabala roditelja. U svakom roditelju odabire se po jedna točka prekida koja označava mjesta na kojima će se dogoditi križanje.

Često, odabir točaka prekida ne odvija se uniformno. Naime, tipični čvorovi stabala imaju prosječni faktor grananja [1] (*eng. branching factor*) jednakim barem 2, što uzrokuje da je većina čvorova nekog stabla u stvari list, odnosno, završni znak. Posljedično, ukoliko bi ovo križanje biralo točke prekida na uniforman način, došlo bi do vrlo male količine razmijenjenog genetskog materijala: nerijetko bi se dogodilo da križanjem nastane nova jedinka koja je gotovo identična jednom od roditelja - od tog roditelja, razlikovala bi se u samo jednom listu koji pripada onom drugom roditelju. Kako bi se razriješio ovaj problem, Koza [2] je predložio kasnije često korišten pristup - prilikom odabira točke prekida, postavlja se vjerojatnost odabira nezavršnog čvora na 0.9, a vjerojatnost odabira lista stabla kao točke prekida na 0.1.

Ovakvim križanjem, dobivaju se dvije različite, nove jedinke. Iako neke implementacije ovog operatora uzimaju oba dva djeteta u novu generaciju, češće je slučaj da se uzima samo jedno dijete. Na slici 3.1 prikazan je primjer jednostavnog križanja.



Slika 3.1: Primjer jednostavnog križanja stabala

Na prvom roditelju (gornji lijevi kut) odabrana točka prekida označena je žutom bojom, a na drugom roditelju zelenom bojom. Na donjem dijelu slike vidljivi su rezultati križanja - dva moguća potomka nastala križanjem ova dva roditelja. Lijevo dijete nastalo je spajanjem podstabla prvom roditelja koje se nalazilo iznad točke prekida i

[1] broj djece pojedinog čvora

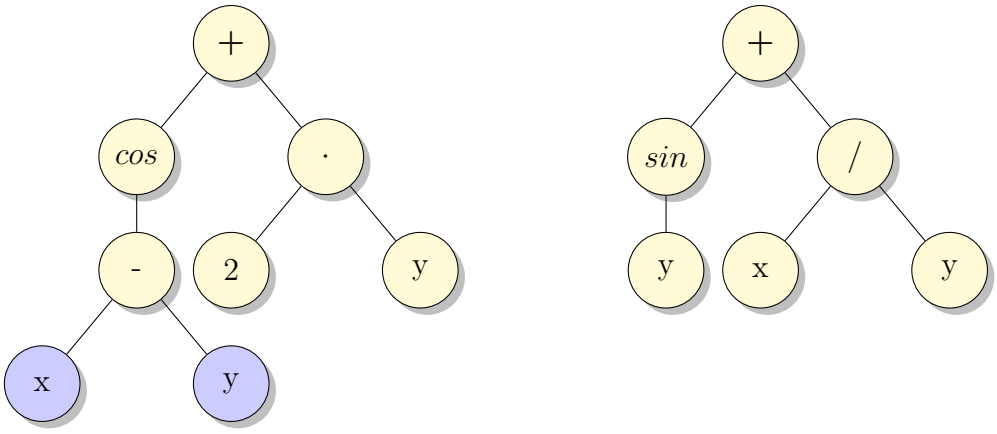
podstabla drugog roditelja koje se nalazilo ispod njegove točke prekida. Desno dijete nastalo je suprotnom kombinacijom - spojeno je podstablo ispod točke prekida prvog roditelja i podstablo iznad točke prekida drugog roditelja.

Budući da je ovaj operator križanja najstariji operator, u daljnjem tekstu uglavnom će služiti kao osnovna usporedba učinkovitosti. Za očekivati je da će svaki drugi operator predstavljen u idućim poglavljima donijeti poboljšanja u odnosu na ovaj operator.

3.2 Križanje s jednom točkom prekida

Tijekom biološke reprodukcije, genetski materijal roditelja rekombinira se na takav način da se određeni geni prenose na otprilike jednako mjesto unutar kromosoma kao što su bili smješteni unutar onog roditeljskog. Ovo se konceptualno razlikuje od načina jednostavnog križanja, gdje je moguće pomicanje podstabala na potpuno drugačiju poziciju od one izvorišne.

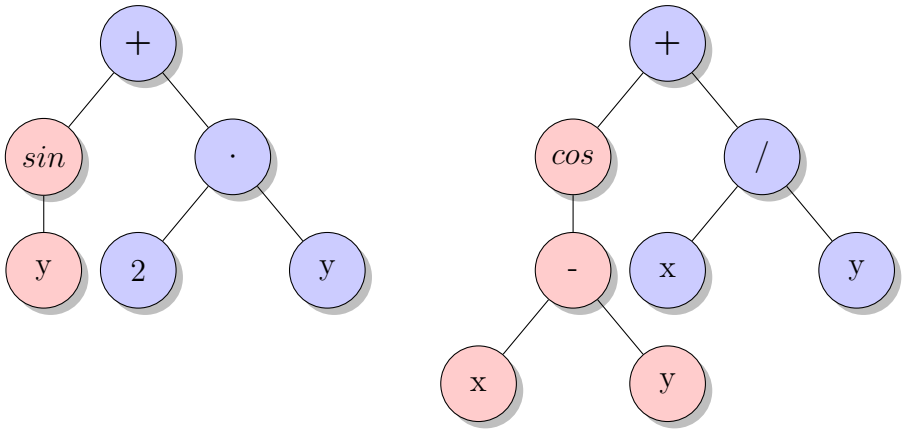
Operatori križanja koji čuvaju poziciju genetskog materijala nazivaju se homolognima. Ovaj operator predstavlja jednu inačicu homolognog križanja i razlikuje se od kasnije spomenutog homolognog operatora križanja. Ovakvi operatori češće su korišteni u linearnom genetskom programiranju gdje je pojam homologije puno intuitivniji i uočljiviji, za razliku od stabala gdje se homolognost može interpretirati na više različitih načina. Posljedično višestrukoj interpretaciji homologije unutar stabala postoji i više vrsta homolognih operatora križanja.



Slika 3.2: Zajedničko područje dva roditelja

Križanje s jednom točkom prekida predstavljeno je u [7], te je prvo križanje koje uzima u obzir homologiju gena. Temelji se na odabiru zajedničke točke križanja unutar roditelja. Pri tome, zajednička točka križanja može se odabrati samo unutar zajedničkih područja roditelja - područja u kojima podstabla oba roditelja imaju potpuno jednak oblik. Primjer zajedničkih područja dan je na slici 3.11, gdje je zajedničko područje dva roditelja označen žutom bojom. Zajednička područja odgovaraju homologiji u smislu da je velika vjerojatnost da podstabla jednakih oblika obavljaju jednaku ili sličnu funkciju. Također, budući da su podstabla jednakog oblika, pozicija određenog gena je intuitivna i jednoznačna.

Na slici 3.3 prikazan je rezultat križanja roditelja sa slike 3.11, gdje je za točku prekida odabran čvor *cos* unutar prvog, odnosno *sin* unutar drugog roditelja.



Slika 3.3: Rezultat križanja roditelja sa slike 3.11

3.2.1 Dosadašnji rezultati

U [8] provedeno je istraživanje o učinkovitosti ovog operatora nad n-paritetnim problemom [2], za $n = 3, 4$. Na slici 3.4 prikazani su rezultati za rješavanje 3-paritetnog problema. Ovdje, stupac *Depth* označava dubinu dobivenog rješenja, p_m vjerojatnost mutacije, te je u ostalim stupcima prikazan napor uložen u izgradnju točnog rješenja s veličinom jedinke u zagradi. Analogno ovome u 3.5 prikazan je jednak scenarij za 4-paritetni problem.

Na slikama je vidljivo kako ovo križanje u prosjeku daje dosta manja rješenja od jednostavnog križanja, no da je do takvih rješenja nešto teže doći. Ovime se može reći kako

[2] rješenje ovog problema je pronaći funkciju n varijabli koja je za parni broj istinitih varijabli istinita

ovo križanje ima prednost nad jednostavnim križanjem - brzina pronalaska konačnog rješenja u genetskom programiranju ionako nije jedan od bitnih faktora koji ukazuje na uspješnost samog algoritma.

Depth	p_m	Normal Crossover	1-pt Crossover	Strict 1-pt Crossover	Mutation Only
4	0	52,000 (45)	308,000 (24)	No Solution	N/A
4	1/256	39,000 (63)	264,000 (27)	810,000 (31)	810,000 (31)
4	1/128	48,000 (38)	110,000 (27)	1,320,000 (31)	396,000 (31)
4	1/64	32,000 (46)	170,000 (26)	315,000 (29)	399,000 (29)
4	1/32	31,000 (42)	128,000 (27)	105,000 (30)	147,000 (31)
4	1/16	54,000 (54)	96,000 (26)	133,000 (28)	86,000 (30)
6	0	24,000 (86)	24,000 (64)	270,000 (88)	N/A
6	1/256	16,000 (88)	27,000 (77)	42,000 (77)	54,000 (92)
6	1/128	15,000 (101)	18,000 (72)	28,000 (75)	44,000 (86)
6	1/64	8,000 (98)	10,000 (81)	8,000 (87)	25,000 (79)
6	1/32	16,000 (100)	14,000 (77)	9,000 (87)	21,000 (92)
6	1/16	19,000 (82)	42,000 (67)	26,000 (67)	28,000 (80)

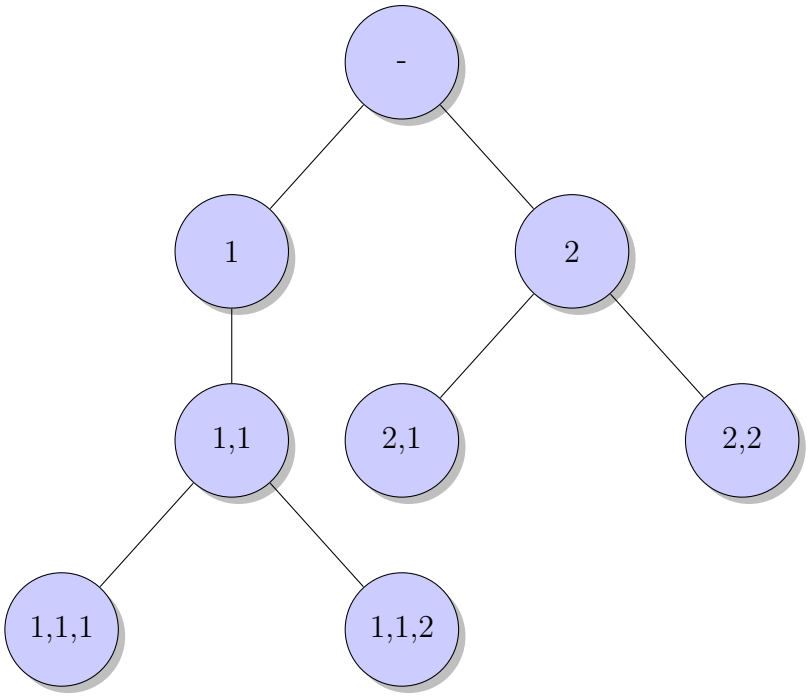
Slika 3.4: Rezultati rješavanja 3-paritetnog problema (preuzeto iz [8])

Depth	p_m	Normal Crossover	1-pt Crossover	Strict 1-pt Crossover	Mutation Only
6	0	1,276,000 (148)	No Solution	No Solution	N/A
6	1/256	238,000 (215)	638,000 (109)	725,000 (119)	880,000 (111)
6	1/128	216,000 (168)	507,000 (98)	357,000 (112)	220,000 (122)
6	1/64	195,000 (154)	224,000 (114)	136,000 (105)	198,000 (116)
6	1/32	611,000 (193)	598,000 (91)	360,000 (111)	510,000 (83)
6	1/16	No Solution	No Solution	No Solution	No Solution
8	0	812,000 (271)	No Solution	No Solution	N/A
8	1/256	126,000 (396)	189,000 (296)	196,000 (313)	319,000 (370)
8	1/128	120,000 (382)	140,000 (296)	129,000 (302)	144,000 (359)
8	1/64	170,000 (377)	144,000 (287)	154,000 (275)	105,000 (210)
8	1/32	1,131,000 (188)	329,000 (112)	500,000 (127)	385,000 (151)
8	1/16	No Solution	No Solution	No Solution	No Solution

Slika 3.5: Rezultati rješavanja 4-paritetnog problema (preuzeto iz [8])

3.3 Križanje s očuvanjem konteksta

Križanje s očuvanjem konteksta [9] također križa dvije jedinke na mjestu točke prekida. Ovaj operator u stablo uvodi koordinate svakog pojedinog čvora. Koordinate označavaju put unutar stabla kojim se dolazi do određenog čvora. Primjer koordinata čvorova stabla dan je na slici 3.6. Ovakve koordinate jednostavno i jednoznačno opisuju poziciju određenog čvora unutar stabla. Primjerice, čvor s koordinatama (1,1,2) nalazi se u drugoj grani podstabla koje se nalazi u prvoj grani prvog podstabla.



Slika 3.6: Koordinate čvorova stabla

3.3.1 Križanje s jakim očuvanjem konteksta

Križanje s jakim očuvanjem konteksta (*eng. strong context preserved crossover - SCPC*) dopušta križanje samo onih podstabala koja imaju potpuno jednake koordinate korijena. Ovakav operator postavlja vrlo stroga ograničenja na proces križanja. U [9] pokazano je kako koristeći ovakav operator, postoji velika vjerojatnost da podstablo generirano na određenoj razini na određenom mjestu stabla nikada ne uspije prijeći u potomke. Ovo bi kroz generacije prouzročilo smanjivanje genetske raznolikosti, što je u potpunoj suprotnosti ulozi operatora križanja. Kako bi se ublažio ovakav utjecaj operatora križanja s jakim očuvanjem konteksta, on se nikada ne koristi ekskluzivno, već u kombinaciji s nekim drugim operatorima križanja.

3.3.2 Križanje sa slabim očuvanjem konteksta

Kako bi se ublažila rigoroznost križanja s jakim očuvanjem konteksta, D'haeseleer [9] je također predložio blažu inačicu ovog križanja - križanje sa slabim očuvanjem konteksta (*eng. weak context preserved crossover - WCPC*). Ova inačica kontekstnog križanja predstavlja pojednostavljenje križanja s jakim očuvanjem konteksta. Korijen

podstabla prvog roditelja koje će sudjelovati u križanju odabire se između skupa čvorova za koje postoji odgovarajući čvor u drugom roditelju (jednako kao i kod križanja s jakim očuvanjem konteksta). Podstablo u drugom roditelju koje će sudjelovati u križanju odabire se tako da mu vršni čvor odgovara vršnom čvoru odabranog podstabla prvog roditelja. Drugim riječima, ako su $T1$ i $T2$ važeći odabiri podstabala za križanje s jakim očuvanjem konteksta, tada su $T1$ i $T2' \subseteq T2$ važeći odabiri podstabala za križanje sa slabim očuvanjem konteksta.

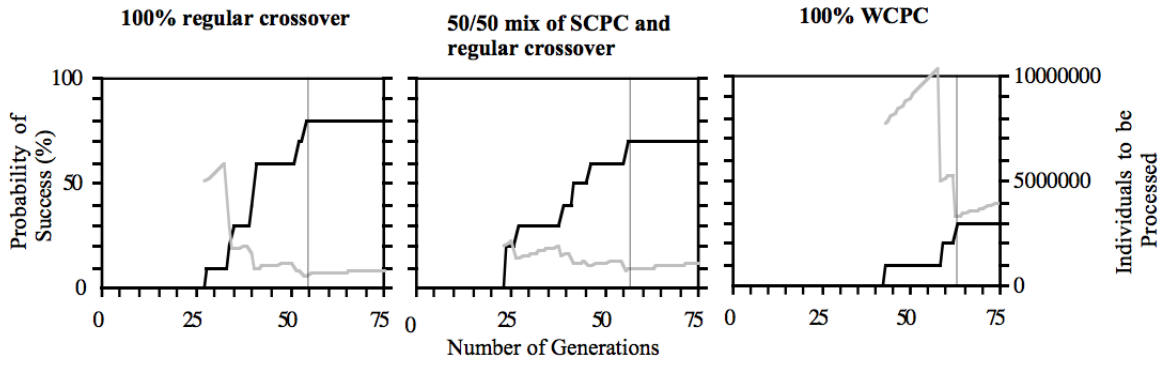
3.3.3 Dosadašnji rezultati

U [9] obavljena su četiri različita eksperimenta koja pokazuju učinkovitost ovih operatora križanja. U nastavku su opisani ti eksperimenti.

Robot sa sposobnosti izmicanja preprekama

Cilj ovog problema je evoluirati robota koji pi prešao što veću površinu zadanog prostora, zaobilazeći pritom prepreke koje mu se nađu na putu. Pri tome, broj pokreta robota je ograničen na neki fiksni broj n.

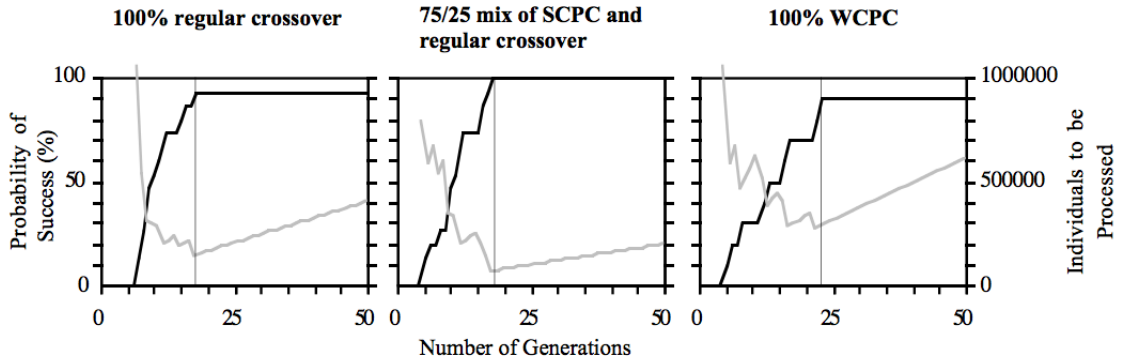
Rezultati ovog eksperimenta pokazali su kako ovo križanje nije dobro za evoluciju rješenja danog problema. Budući da evoluirani robot samo jednom izvršava svoj program predstavljen stablom, to stablo bi trebalo biti veliko i opisivati svaki korak robota. Budući da ova križanja, a pogotovo križanje s jakim očuvanjem konteksta ne potiče prijenos krajnjih podstabala rješenja, dobiven rezultat bio je i za očekivati. Na slici 3.7 dana je usporedba učinkovitosti uobičajenog (jednostavnog) operatora križanja, kombinacije jednostavnog i križanja s jakim očuvanjem konteksta (u omjeru 1:1) i križanja s jakim očuvanjem konteksta. Vidljivo je kako je jednostavno križanje superiorno nad kontekstnim križanjima za rješavanje ovog problema.



Slika 3.7: Učinkovitost različitih operatora križanja za problem robota sa sposobnošću izmicanja prepreka. lijevo: jednostavno križanje, sredina: kombinacija jednostavnog i križanja sa slabim očuvanjem konteksta, desno: križanje s jakim očuvanjem konteksta (preuzeto iz [9])

Iterirana inačica robota sa sposobnosti izmicanja preprekama

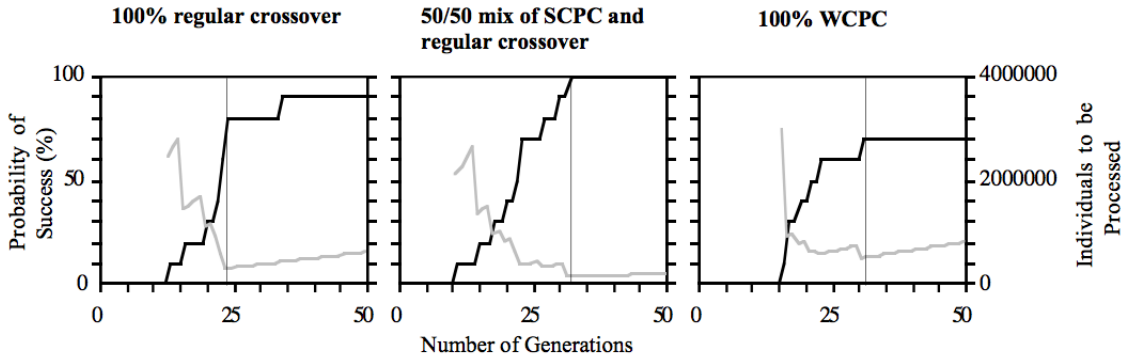
Za razliku od prethodno opisanog problema, gdje se program predstavljen stablom izvršava samo jedan put, u ovom problemu unutar dopuštenog broja koraka dopušteno je izvoditi program po nekoliko puta. Ovakvo ponašanje u stvari je i prirodnije od jednostrukog izvršavanja stabla, i očekivano je da će u ovom slučaju kontekstni operatori djelovati efikasnije nego kod rješavanja prethodnog problema. Na slici 3.8 prikazana je usporedba učinkovitosti operatora križanja za ovaj problem. Vidljivo je kako je kombinacija jednostavnog i križanja s jakim očuvanjem konteksta (u omjeru 1:3) djelovala najbolje pri rješavanju ovog problema.



Slika 3.8: Učinkovitost različitih operatora križanja za iterativni problem robota sa sposobnošću izmicanja prepreka. lijevo: jednostavno križanje, sredina: kombinacija jednostavnog i križanja sa slabim očuvanjem konteksta, desno: križanje s jakim očuvanjem konteksta (preuzeto iz [9])

11 - multipleksor

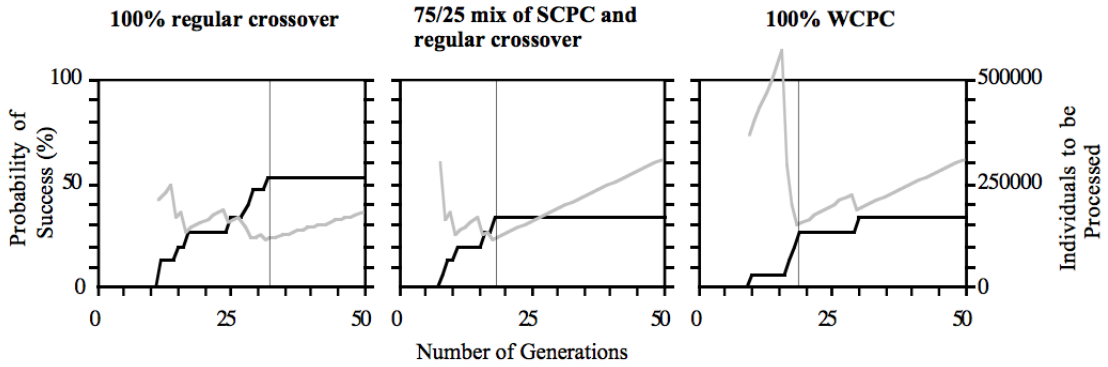
Cilj boolean 11 - multipleksor problema je pronaći logičku funkciju koja za 3 predana adresna bita daje podatak veličine 8 bita. U ovom problemu, kombinacija jednostavnog i križanja s jakim očuvanjem konteksta (u omjeru 1:1) pokazala se daleko boljim odabirom za operator križanja nego jednostavno križanje. Križanje sa slabim očuvanjem konteksta se i u ovom eksperimentu pokazalo prilično inferiorno ostalim dvama operatorima. Na slici 3.9 prikazana je usporedba učinkovitosti različitih operatora križanja na ovaj problem.



Slika 3.9: Učinkovitost različitih operatora križanja za problem boolean 11 - multipleksora. lijevo: jednostavno križanje, sredina: kombinacija jednostavnog i križanja sa slabim očuvanjem konteksta, desno: križanje s jakim očuvanjem konteksta (preuzeto iz [9])

Centralno sakupljalište hrane

Efikasno rješenje ovog problema je evoluirani program koji, kada je pokrenut nad svakim umjetnim mravom kolonije, prouzrokuje transport sve hrane s danog područja na jedno, centralno mjesto. Dobro rješenje ovog problema mora uključivati i interakciju i kooperaciju između mrava. Radi toga, evoluirani program najčešće su kratki, sadržavajući samo nekoliko koraka u svakoj iteraciji izvođenja. Ovo upućuje na činjenicu da bi kontekstno križanje moglo biti vrlo korisno za evoluciju takvog, efikasnog rješenja. Na slici 3.10, jednako kao i za prethodne eksperimente, prikazana je usporedba učinkovitosti različitih operatora križanja na ovaj problem.



Slika 3.10: Učinkovitost različitih operatora križanja za problem centralnog sakupljališta hrane. lijevo: jednostavno križanje, sredina: kombinacija jednostavnog i križanja sa slabim očuvanjem konteksta, desno: križanje s jakim očuvanjem konteksta (preuzeto iz [9])

Ova istraživanja pokazala su kako križanje s očuvanjem konteksta nije učinkovito kada je upotrebljeno samo, već je efikasnije u kombinaciji s nekim drugim operatorom. Kombinacija jednostavnog križanja i križanja s jakim očuvanjem konteksta pokazala se kao vrlo učinkovit operator za iterirane programe koji predstavljaju neovisnu jedinku i za pronalazak logičkih funkcija.

3.4 Križanje pravedno s obzirom na veličinu

Česta pojava u genetskom programiranju je prekomjeran rast jedinki kako teku generacije (*eng. bloat*). Postoji više načina rješavanja ili sprječavanja ovog problema. Jedan od načina kako riješiti ovaj problem nad već generiranim jedinkama je podrezivanje (*eng. pruning*), odnosno odstranjivanje onih dijelova stabla koji su izvan dopuštene dubine. Drugi način je da se rezultat operatora križanja odbacuje sve dok je stablo producirano takvim križanjem veće dubine nego što je dopušteno. Moguće je i kažnjavati predugačka rješenja umanjivanjem njihove dobrote. Ovaj operator ovaj problem rješava u samom postupku križanja. Naime, kontrolirajući veličinu podstabala koja ulaze u križanje, direktno kontrolira veličinu novogeneriranog potomka.

Nakon što je algoritam dosegnuo stagnaciju u napredovanju, uobičajeni operatori križanja pokušat će proširiti prostor pretrage. No, radi pojave prekomjernog rasta, ova potraga više će se protezati u prostoru duljine samog rješenja nego u prostoru različitih rješenja.

Kod ovakvog pristupa, postoji veća vjerojatnost za preživljavanjem dijelova rješenja koji nisu štetni, ali nisu ni korisni - podržavajući tako daljnji rast jedinki. Kod križanja pravednog s obzirom na veličinu, to nije slučaj - nakon dosegnutog platoa, ono će istraživati različita rješenja, čisto iz razloga što je jedinka dobivena ovakvim križanjem otprilike jednake veličine kao i svoji roditelji.

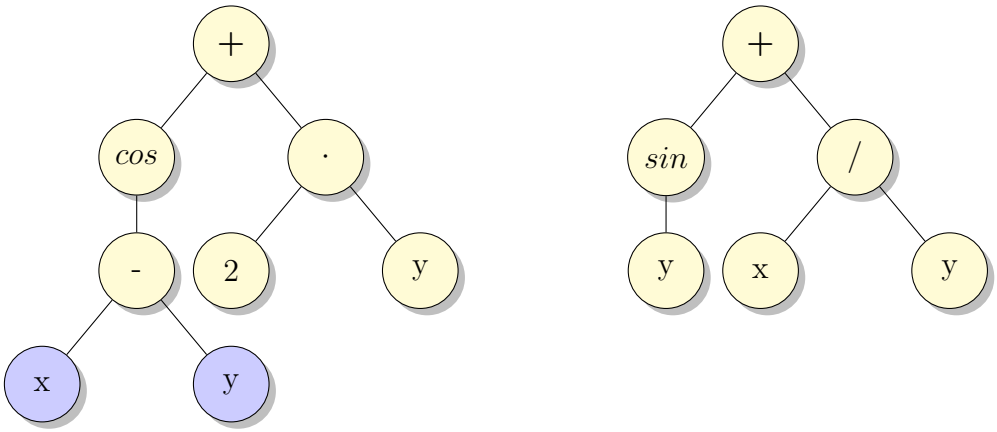
U [10] predstavljeno je križanje pravedno s obzirom na veličinu. Odabir točke prekida u prvom roditelju koji sudjeluje u križanju obavlja se nasumično. Pri tome, pristranost odabiru nezavršnog čvora stabla nad odabirom lista stabla iznosi 0.9. Ono po čemu se ovo križanje razlikuje od jednostavnog križanja je odabir točke prekida u drugom roditelju. Kako bi se pronašla druga točka prekida, izračunava se veličina stabla koje će biti obrisano u djetetovoj kopiji prvog roditelja. Nakon toga, u drugom stablu pronalaze se podstabla veličine ne veće od $1 + 2 \cdot |veličina\_obrisanog\_podstabla|$. Ovime se osigurava da novostvorena jedinka neće biti veća od $veličina\_prvog\_roditelja + |veličina\_obrisanog\_podstabla| + 1$.

3.4.1 Dosadašnji rezultati

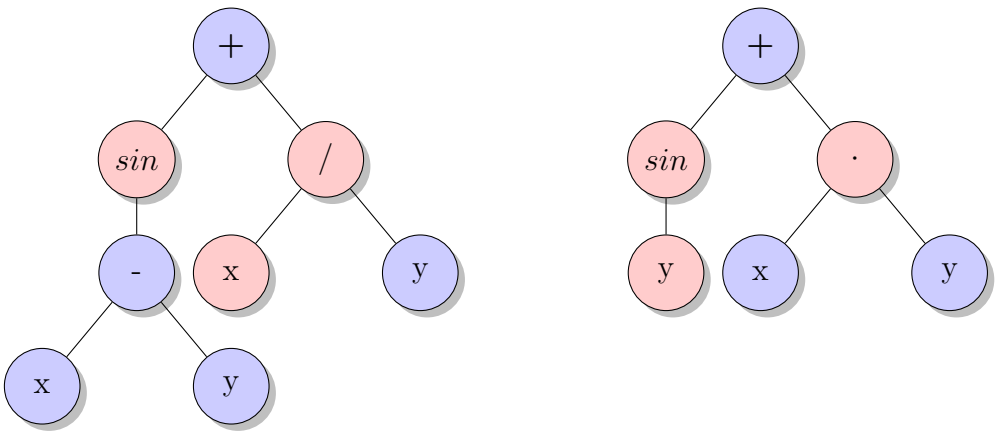
U [10] provedeno je istraživanje nad četiri različita problema; po dva za simboličku regresiju (polinomi petog i šestog stupnja) i po dva za logičke funkcije (6 i 11 - multiplekser). Pokazano je kako je ovaj operator križanja kroz generacije proizvodio značajno, pa čak i do deset puta manje jedinke nego jednostavni operator križanja. Uočeno je da je u slučaju kada se početna populacija sastoji od jedinki koje su puno manje nego što je potrebno, jednostavno križanje i dalje superiornije pri pronalasku rješenja. Ovo se može pokazati kao značajan nedostatak ukoliko se na početku algoritma ne poznaje okvirna veličina dobrog rješenja. Dokazano je da ovaj operator značajno smanjuje prekomjeran rast jedinki. Za usporedbu, prosječan rast jedinki tijekom generacija za jednostavno križanje je linearno, dok je prosječan rast jedinki za ovo križanje subkvadratno.

3.5 Uniformno križanje

Uniformno križanje predstavljeno je u [11]. Inspirirano je uniformnim križanjem nestablastih struktura u genetskom algoritmu. U genetskom algoritmu, ovo križanje (budući da su sve jedinke jednakih dužina) prolazi kroz oba roditelja te na svakoj poziciji nasumično odabire gen iz prvog ili drugog roditelja te ga ugrađuje u dijete. Kada bi u genetskom programiranju sve jedinke bile potpuno jednakog oblika, ovo križanje funkcioniralo bi potpuno jednako. No, budući da je to vrlo rijedak slučaj, potrebno je, kao i u križanju s jednom točkom prekida, pronaći zajedničko područje dva roditelja. Nakon što je to područje pronađeno, s lakoćom se smiju izmjenjivati čvorovi roditeljskih stabala - no pritom je potrebno paziti na broj djece koji taj čvor posjeduje. Na slici 3.12 prikazan je primjer jednog mogućeg rezultata uniformnog križanja.



Slika 3.11: Zajedničko područje dva roditelja



Slika 3.12: Mogući rezultati križanja roditelja sa slike 3.11

Velika prednost ovog križanja je velika stopa razmjene genetskog materijala. Budući

da se čvorovi koji će završiti u djetetu biraju uniformno, s omjerima vjerojatnosti 1:1, velika je vjerojatnost da će dijete sadržavati 50% genetskog materijala prvog roditelja i 50% genetskog materijala drugog roditelja. Ovo je vrlo pogodno za brzo napredovanje algoritma.

3.5.1 Dosadašnji rezultati

Autor ovog križanja proveo je eksperimente nad paritetnim problemom 4 varijable. Cilj ovog problema je pronaći logičku funkciju koja je istinita za parni broj istinitih varijabli. Tablica istinitosti za ovaj problem prikazana je na tablici ispod (3.1).

A	B	C	D	P
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Tablica 3.1: Tablica istinitosti za paritetni problem 4 varijable

Eksperimentima je dokazano kako za razliku od jednostavnog križanja i križanja s jednom točkom prekida, u kojima je prosječna stopa razmjene genetskog materijala

oko 5%, ovo križanje uistinu ima konstantnu stopu razmjene genetskog materijala od oko 50%.

Pokazano je kako jednostavno križanje pretragu prostora rješenja odvija lokalno, nasljeđujući većinu genetskog materijala od jednog roditelja. Jednostavno križanje nije idealno za brzo pretraživanje prostora - bolje je za fino ugođavanje prilično dobrog rješenja. Osim toga, jednostavno križanje vrlo lako zapne u nekom od lokalnih optimuma.

Križanje s jednom točkom prekida bolje se nosi s veličinom prostora pretrage, no nakon nekog vremena počinje sagledavati sve manji dio prostora - lokalizirajući tako pretragu s povećanjem vjerojatnosti zapinjanja u lokalnom optimumu.

Uniformno križanje prelazi preko ovih problema. Za razliku od jednostavnog križanja, ono nije pristrano na križanje listova ili podstabala. Ovime omogućuje slobodniju i raznolikiju izmjenu genetskog materijala između roditelja, ubrzavajući tako konvergenciju rješenja u globalni optimum. Također, pokazano je kako nakon nekog vremena, sve jednike populacije budu otprilike jednake veličine, što kod upotrebe jednostavnog križanja i križanja s jednom točkom prekida nije slučaj. Osim navedenih svojstava, nije uočena prevelika razlika u samoj uspješnosti križanja da pronađe dobro rješenje problema - rad se pretežito fokusirao na razliku u količini izmijenjenog genetskog materijala.

3.6 Homologno križanje

Homologno križanje [10] temelji se na prirodnom svojstvu križanja, homologiji. Homologija u prirodi osigurava da se križanje uvijek događa između organizama koji imaju uglavnom identične sekvence gena unutar kromosoma. Ovime se podupire nedestruktivno križanje - kombiniraju se geni koji rade točno određenu ulogu, gdje je uloga uvjetovana pozicijom samog gena.

Homologno križanje, u smislu križanja u genetskom programiranju, koristi pozicijsku homologiju. Pozicijska homologija potiče izmjenu genetskog materijala samo ako se ono događa na istoj, ili dovoljno bliskoj poziciji unutar dva genoma.

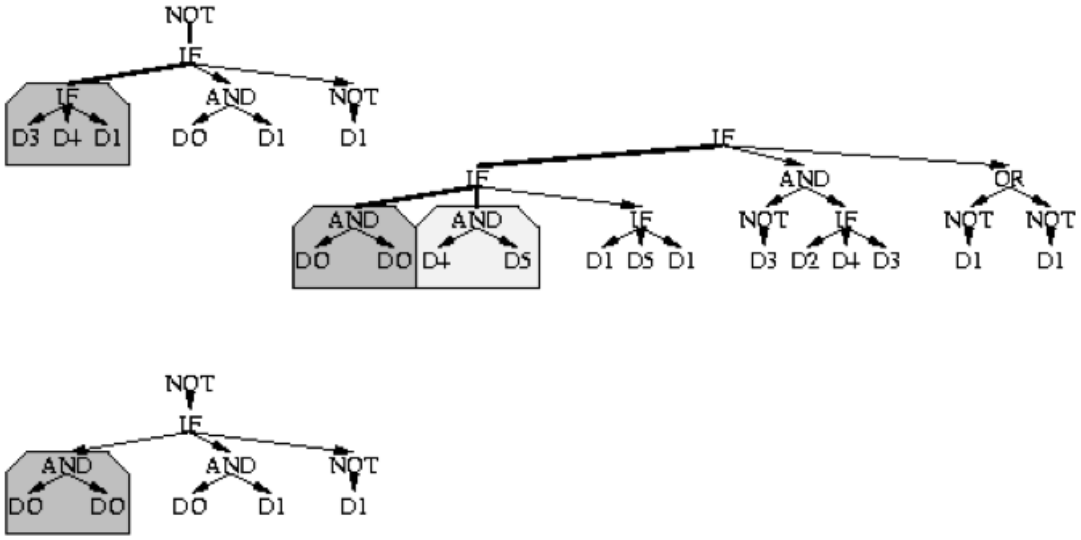
Uobičajeni operatori križanja, prilikom odabira podstabala ili čvorova koji sudjeluju u križanju uglavnom ne uzimaju u obzir njihovu poziciju. Ne uzimajući to u obzir, ne-

rijetko dolazi do naglog rasta jedinki svakom novom generacijom (*eng. bloat*). Budući da ovi operatori najčešće ne uzimaju u obzir prekomjeran rast jedinki koje proizvode, potrebno je modificirati sam algoritam tako da se prevelike jedinke kažnjavaju manjom dobrotom. Time, iako možda novonastala jedinka izvrsno rješava zadan problem, može se dogoditi da bude odbačena zbog svoje veličine. Iz razloga što se ne uzimaju u obzir pozicije niti veličine podstabala koja sudjeluju u križanju, mala je šansa da će novonastala jedinka biti u okviru željene veličine, povećavajući šansu da će u budućnosti zbog te veličine i izumrijeti. Drugi razlog zbog kojega prekomjeran rast jedinke nije dobar, a usko je povezan s prethodnim, je taj da, u prevelikoj jedinci vjerojatno postoji puno nepotrebnog ponašanja - ponašanje koje nije potrebno, a kroz generacije je preživjelo samo zato što ne donosi nikakvu štetu.

Drugi problem koji se javlja kod uobičajenih operatora križanja je taj da ti operatori potpuno nasumično prenose dijelove programa u novu jedinku. Budući da je taj fragment jedinke preživio proces selekcije, za pretpostaviti je da postoji velika šansa da on predstavlja relevantan dio za put do konačnog rješenja. Također, velika je šansa da korisnost istog fragmenta uvelike ovisi i o kontekstu u kojem se on izvršava. Stavljajući takav fragment na nasumično mjesto u novu jedinku donosi šansu da će taj kontekst biti uništen.

Kako bi se ovi problemi riješili prilikom samog križanja, a time se i povećale šanse za kvalitetu i nastavak života potomaka, homologni operator križanja prilikom samog križanja uzima u obzir i veličinu i poziciju podstabala roditelja koja će se iskrižati. Ovime se u samoj srži operatora sprječava prekomjeran rast potomaka i s većom vjerojatnošću nego inače, podržava prijenos konteksta pojedinih fragmenata rješenja u ono novonastalo.

Ovo križanje u stvari je inačica križanja pravednog s obzirom na veličinu. Odabir potencijalnih podstabala drugog roditelja potpuno je jednak u ovom križanju kao i kod križanja pravednog s obzirom na veličinu. Razlika u tome je ta da se, između svih potencijalnih podstabala drugog roditelja koji bi mogli sudjelovati u križanju, odabire ono podstablo koje je najbliže točki prekida prvog roditelja. Na slici 3.13 prikazan je jedan takav primjer križanja - od dva potencijalna podstabala drugog roditelja, za križanje je odabrano ono koje je bliže točki prekida u prvom roditelju.



Slika 3.13: Primjer homolognog križanja (preuzeto iz [10])

3.6.1 Dosadašnji rezultati

U [10] provedeno je istraživanje nad četiri različita problema; po dva za simboličku regresiju (polinomi petog i šestog stupnja) i po dva za logičke funkcije (6 i 11 - multipleksor). Pokazano je kako homologno križanje po sposobnosti pronalaska rješenja danih problema nije značajno bolje od križanja pravednog s obzirom na veličinu. No, kao i kod križanja pravednog s obzirom na veličinu, pokazano je kako je ovo križanje vrlo učinkovito prilikom sprječavanja pojave prekomjernog rasta jedinki.

Razlog sličnosti u rezultatima s križanjem pravednim s obzirom na veličini može biti prouzrokovano različitim razlozima. Jedan od razloga je mogućnost da kod ovog operatora u nekim slučajevima nije postojao izbor podstabla koji bi bio u skladu s homologijom, te bi operator bio primoran odabrati podstablo koje ne odgovara u potpunosti načelu odabira. Postojanje ovog slučaja pokazano je u spomenutom radu.

3.7 Deterministično križanje

Deterministično križanje (*eng. deterministic functional crossover*), opisano u [12], ostvaruje križanje dvije jedinke na osnovi funkcijske sličnosti. Ovaj operator primjenjiv

je samo na probleme simboličke regresije.

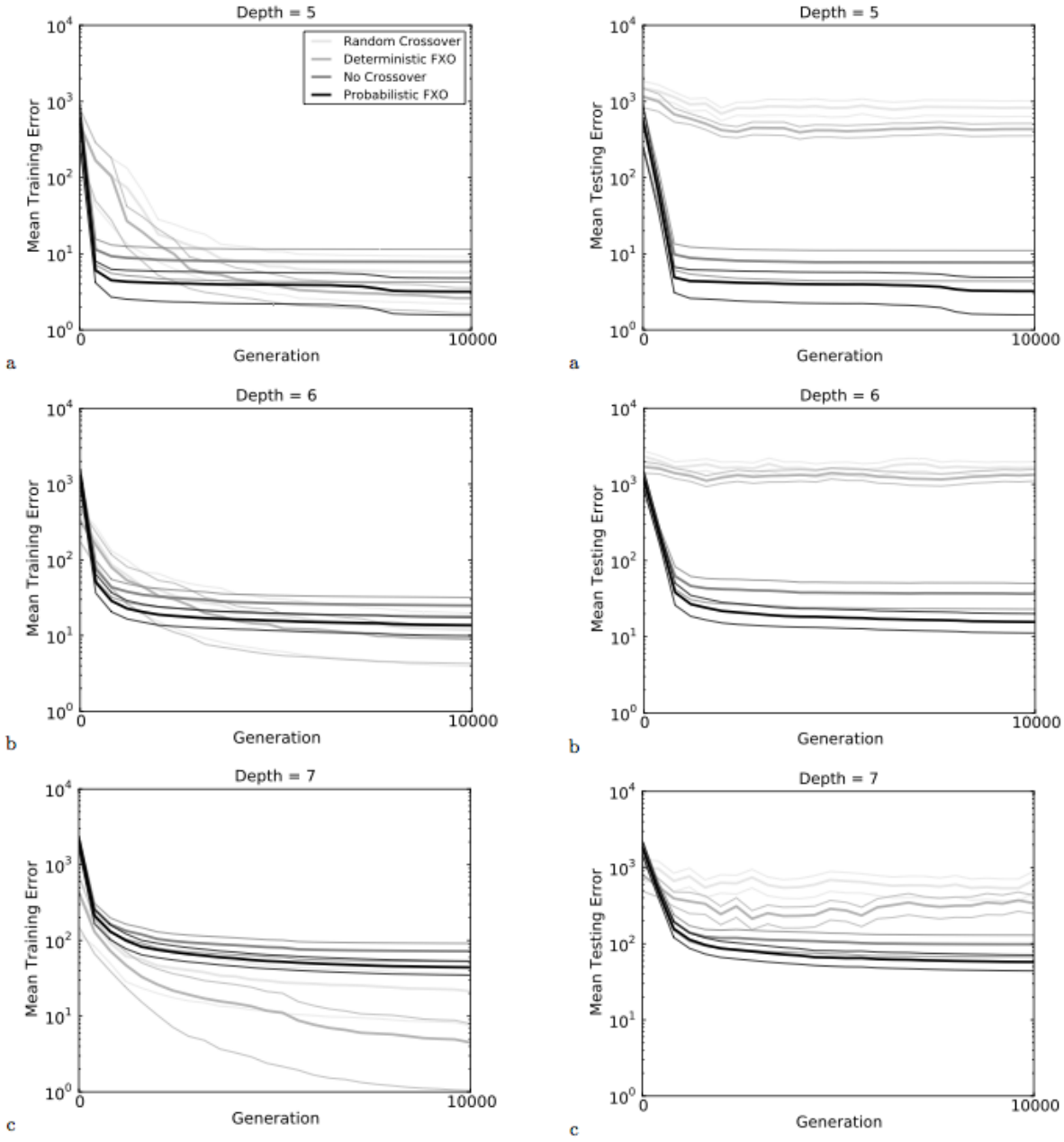
Kako bi se funkcijska sličnost dva gena, odnosno čvorova stabla koji izgrađuju jedinku, mogla odrediti, za svaki gen zapisuje se minimalna i maksimalna vrijednost koju može poprimiti za podatke iz skupa za učenje. Funkcijska sličnost (u radu nazvana funkcijskom udaljenošću) između dva gena i i j, računa se kao:

$$d_{i,j} = \frac{1}{2}(|max_i - max_j| + |min_i - min_j|) \tag{3.1}$$

U radu je opisan glavni nedostatak ovog križanja - postoji velika vjerojatnost da križanje koje se dogodi pod utjecajem ovog operatora bude neutralno, odnosno, da se novonastala jedinka uopće ne razlikuje od jednog od roditelja. Ovaj slučaj dogodi se kada se u prvom roditelju izabere podstablo koje je strukturno jednako odabranom podstablu drugog roditelja (može biti interpretirano na jednak način nakon što se izraz predstavljen podstablom simplificira). Križanjem takvih podstabala, originalno ponašanje se uopće ne mijenja te je rezultantna jedinka potpuno jednaka svome roditelju. Pojava neutralnog križanja to je češća što jedinke sve više i više rastu (čime se povećava vjerojatnost odabira strukturno jednakog podstabla). Kako bi se riješio ovaj problem, u istom radu predloženo je probabilističko križanje, koje je opisano u idućem poglavlju.

3.7.1 Dosadašnji rezultati

Na slici 3.14 prikazani su grafovi usporedbe učinkovitost deterministčkog križanja u usporedbi s jednostavnim i probabilističkim križanjem i slučaja kada nije upotrijebljen niti jedan operator križanja. Pri tome, grafovi na lijevoj strani pokazuju performanse dobivenih rješenja na skupu za učenje, a oni na desnoj strani, performanse na skupu za testiranje. Eksperimenti su bili provedeni na primjerima simboličke regresije. Na desnoj strani slike, vidljivo je kako ne postoji prevelika razlika između učinkovitosti danih operatora.



Slika 3.14: Usporedba učinkovitosti determinističkog, jednostavnog i probabilističkog križanja i slučaja kada nije upotrijebljen niti jedan operator križanja (preuzeto iz [12])

Također, vidljivo je kako je ovo križanje dalo superiorno rješenje na skupu za učenje naspram ostalima za slučaj kada je dubina stabla jednaka 7. U suprotnosti, odnosno, posljedično tome, na skupu za testiranje isti slučaj dao je značajno slabije rezultate što upućuje na prenaučavanje jedinki.

Također, pokazano je da ovo križanje ne podržava prekomjeran rast jedinki tijekom generacije. Dapače, dana rješenja često su ispala nešto manja od onog ciljanog.

3.8 Probabilističko križanje

Kako bi se riješio problem neutralnog križanja koji se pojavio kod deterministčkog križanja, u [12] je predloženo probabilističko križanje (eng. probabilistic functional crossover). Jednako kao i kod deterministčkog križanja, za svaki čvor u stablu zapisuje se minimalna i maksimalna vrijednost čvora za podatke iz skupa za učenje (izraz 3.1).

Prilikom križanja, nakon što se u prvom stablu nasumično odabere točka križanja izračunava se funkcionalna udaljenost između njega i svakog čvora unutar drugog roditelja (jednako kao i kod deterministčkog križanja). Nakon što su dobivene sve moguće udaljenosti, one se normaliziraju prema 3.2:

$$d'_{i,j} = \frac{d_{i,j}}{\sum_{k=1}^{s} d_{i,k}} \tag{3.2}$$

Normalizacija je pogodna iz razloga što se s najvećom vjerojatnošću želi odabrati funkcionalno najbliži čvor. Ovako dobivene normalizirane udaljenosti potom se invertiraju i ponovno normaliziraju u odnosne na te invertirane vrijednosti prema 3.3:

$$p_{i,j} = \frac{1 - d'_{i,j}}{\sum_{k=1}^{s}(1 - d'_{i,k})} \tag{3.3}$$

Time dobivamo to veću vjerojatnost što je udaljenost između dva čvora manja. Nakon ovih operacija, nasumično, ali s izračunatim vjerojatnostima, odabire se podstablo unutar drugog roditelja koje će se iskoristiti za križanje. Pokazano je kako ovakvo križanje pridonosi većem postotku korisnog križanja, a time i bržoj konvergenciji nego uobičajeni operatori, iako je priznato kako je ovakav izračun udaljenosti prilično gruba aproksimacija funkcionalne udaljenosti.

3.8.1 Dosadašnji rezultati

Jednako kao i za deterministčko križanje, provedeni eksperimenti za ovo križanje uključuju usporedbu probabilističkog, deterministčkog i jednostavnog križanja i slučaja

kada nije upotrijebljen niti jedan operator križanja. Rezultati su prikazani su na slici 3.14. Vidljivo je kako probabilističko križanje konstantno bolje djeluje od jednostavnog križanja i slučaja kada se ne koristi niti jedan operator križanja.

Osim toga, dokazano je kako je korisnost križanja [3], kada se koristi probabilistički operator križanja, puno veća nego u ostalim slučajevima. Pokazano je kako se ovim križanjem dobivaju jedinke koje su značajno veće od točnog rješenja. Naravno, pitanje je kako interpretirati ovu pojavu, budući da se izraz koji predstavlja jedinka gotovo uvijek može pojednostavniti čime bi stablo bilo puno manje.

Uočena je korelacija između veličine roditelja i događanja korisnog križanja. No, to je i za očekivati iz razloga što je omjer broja listova na prema unutarnjim čvorovima to veći što je samo stablo veće. Radi toga, vjerojatnost odabira lista za ulazak u križanje se povećava, čime se i smanjuje mogućnost za destruktivnim križanjem.

3.9 Semantičko križanje

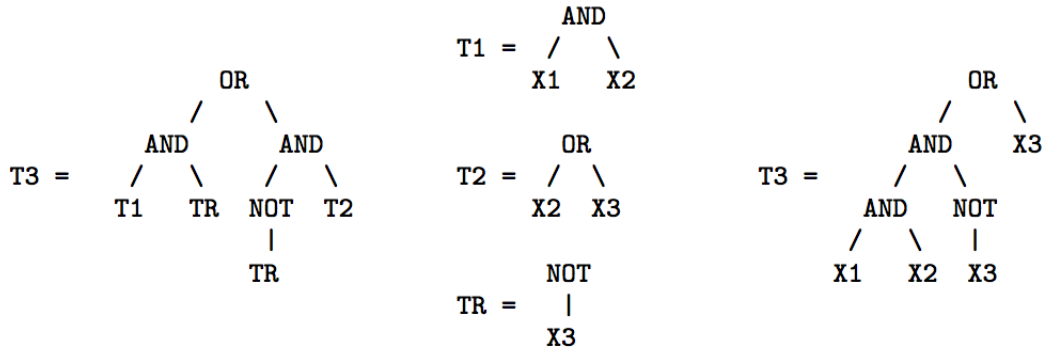
Suprotstavljeno tradicionalnim operatorima križanja u genetskom programiranju koji ignoriraju semantiku samog programa, u [13] opisano je semantičko križanje. Ovo križanje razlikuje nekoliko inačica, od kojih je svaka primjenjiva na određenu vrstu problema. Semantičko križanje različito je definirano za situacije kada stablo jedinke predstavlja logičku ili realnu funkciju, ili ako je jedinka program.

Ukoliko jedinka predstavlja logičku funkciju, semantičko križanje definirano je kao:

$$(T1 \wedge T2) \vee (\overline{TR} \wedge T2), \tag{3.4}$$

gdje T1 i T2 predstavljaju roditelje koji sudjeluju u križanju, a TR predstavlja nasumično generirano stablo. Na slici 3.15 prikazana je shema semantičkog križanja za logičke funckije.

[3] križanje se smatra korisnim ukoliko producira jedinku koja ima dobrotu bolju od oba roditelja



Slika 3.15: Lijevo: shema semantičkog križanja za logičke funkcije; sredina: roditelji i nasumično generirana jedinka; desno: rezultat križanja roditelja prikazanih u sredini (preuzeto iz [13])

Analogno ovome, za realne funkcije, odnosno, simboličku regresiju, semantičko križanje predstavljeno je sljedećim izrazom:

$$(T1 \cdot T2) + ((1 - TR) \cdot T2). \tag{3.5}$$

Za slučaj kada jedinka predstavlja program, semantičko križanje definira se kao:

$$\mathbf{IF}(CONDR)\mathbf{THEN}(T1)\mathbf{ELSE}(T2), \tag{3.6}$$

gdje je CONDR nasumično generiran program čiji je izlaz interpretiran kao logička vrijednost.

Iz ovoga je jasno vidljiv najveći nedostatak ovog operatora - prekomjeran rast novonastalih jedinki. Naime, prilikom svakog križanja stablo će se sigurno uvećati po dubini za barem jednu razinu. Ovo može postati znatan problem iz razloga što je rast vrlo brz, te se često nakon nekog vremena dogodi to da za novodobiveno stablo ne postoji prostor za pohranu. Radi ovoga, nakon svakog križanja, potrebno je pojednostavniti novonastalu jedinku. Ovo pojednostavljenje naravno mora osigurati da je funkcionalnost jedinke nakon pojednostavljenja ostala nepromijenjena.

Glavna prednost ovog semantičkog operatora je činjenica da je, za razliku od nekih drugih predloženih semantičkih operatora, primjenjiv na širok skup problema.

3.9.1 Dosadašnji rezultati

Eksperimenti u [13] provedeni su nad sve tri vrste problema za koje je ovaj operator definiran; logičke funkcije, simboličku regresiju i programe. Ovo istraživanje je usporedilo genetsko programiranje (GP) sa semantičkim genetskim programiranjem (*eng. semantic genetic programming - SGP*) i semantičkim stohastičkim planinskim penjačem (*semantic stochastic hill climber - SSHC*). Također, napravljena je usporedba s algoritmom genetskog programiranja koje se vrti jednako dugo kao i SGP i SSHC za pojedini problem (*GPt*). U nastavku su opisani dobiveni rezultati za pojedinu vrstu problema.

Logičke funkcije

Učinkovitost logičkog semantičkog operatora križanja provedena je na 5 različita problema, zajedno s nekoliko inačica za svaki problem. Problemi su redom: n -komparator (6, 8 i 10), n -multipleksor (6 i 11), n -paritet (5, 6, 7, 8, 9 i 10), nasumično generirana funkcija n varijabli (5, 6, 7, 8, 9, 10 i 11) i n-istinita funkcija [4] (5, 6, 7 i 8). Dobrota je računata kao Hammingova udaljenost vrijednosti stvarne funkcije i dobivenog rješenja. Na slici 3.16 prikazana je tablica rezultata.

Stupac *Hits %* predstavlja postotak uspješnosti nad skupom za učenje za najbolju jedinku u populaciji, za 30 pokretanja algoritma. Pri tome *avg* predstavlja prosjek, a *sd* standardnu devijaciju. U stupcu *Length* prikazani su logaritmi po bazi 10 veličine dobivenih rješenja. Valja napomenuti da je nakon upotrebe semantičkog križanja primijenjena simplifikacija dobivenog rješenja, iz razloga što bi veličina rješenja eksponencijalno rasla u suprotnom. Vidljivo je kako SSHC i SGP konstantno pronalaze bolja rješenja od GP-a sa semantičkim križanjem, no GP proizvodi manje jedinke (čemu je glavni razlog pojednostavljenje izraza koje se u ostalim algoritmima ne obavlja - što umanjuje relevantnost ovog zaključka). Vidljivo je kako GPt pokazuje bolje performanse od GP-a, no još uvijek se pokazuje lošijim od SGP-a i SSHC-a.

[4] n-istinita funkcija je funkcija n varijabli koja je za svaku kombinaciju varijabli istinita

Problem	Hits %								Length			
	GP		GPt		SSHC		SGP					
	avg	sd	avg	sd	avg	sd	avg	sd	GP	GPt	SSHC	SGP
Comparator6	80.2	3.8	90.9	3.5	99.8	0.5	99.5	0.7	1.0	2.0	2.9	2.8
Comparator8	80.3	2.8	94.9	2.4	100.0	0.0	99.9	0.2	1.0	2.3	2.9	3.0
Comparator10	82.3	4.3	95.3	0.9	100.0	0.0	100.0	0.1	1.6	2.4	2.7	3.0
Multiplexer6	70.8	3.3	94.7	5.8	99.8	0.5	99.5	0.8	1.1	2.2	2.7	2.9
Multiplexer11	76.4	7.9	88.8	3.4	100.0	0.0	99.9	0.1	2.2	2.4	2.9	2.6
Parity5	52.9	2.4	56.3	4.9	99.7	0.9	98.1	2.1	1.4	1.7	2.9	2.9
Parity6	50.5	0.7	55.4	5.1	99.7	0.6	98.8	1.7	1.0	1.9	3.0	3.0
Parity7	50.1	0.2	51.7	2.8	99.9	0.2	99.5	0.6	1.0	1.7	3.0	3.1
Parity8	50.1	0.2	50.6	0.9	100.0	0.0	99.7	0.3	1.0	1.6	3.4	3.4
Parity9	50.0	0.0	50.2	0.1	100.0	0.0	99.5	0.3	1.0	1.3	3.8	3.8
Parity10	50.0	0.0	50.0	0.0	100.0	0.0	99.4	0.2	0.9	1.2	4.1	4.1
Random5	82.2	6.6	90.9	6.0	99.5	1.2	98.8	2.1	0.9	1.6	2.7	2.8
Random6	83.6	6.6	93.0	4.1	99.9	0.4	99.2	1.3	1.2	1.9	2.9	2.8
Random7	85.1	5.3	92.9	3.8	99.9	0.2	99.8	0.4	1.1	2.0	2.8	2.9
Random8	89.6	5.3	93.7	2.4	100.0	0.1	99.9	0.2	1.4	2.0	3.0	2.9
Random9	93.1	3.7	95.4	2.3	100.0	0.1	100.0	0.1	1.5	1.8	2.9	2.9
Random10	95.3	2.3	96.2	2.0	100.0	0.0	100.0	0.0	1.5	1.8	2.8	3.0
Random11	96.6	1.6	97.3	1.5	100.0	0.0	100.0	0.0	1.6	1.7	2.7	3.1
True5	100.0	0.0	100.0	0.0	99.9	0.6	100.0	0.0	1.1	1.3	2.0	2.4
True6	100.0	0.0	100.0	0.0	99.8	0.6	100.0	0.0	1.2	1.2	2.6	2.5
True7	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0	1.2	1.2	2.9	2.6
True8	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.1	1.2	1.4	3.3	2.9

Slika 3.16: Usporedba učinkovitosti GP-a i GPt-a sa SGP-om i SSHC-om (preuzeto iz [13])

Simbolička regresija

Za ocjenu učinkovitosti ovog operatora na simboličkoj regresiji korišteni su različiti polinomi, od trećeg do desetog stupnja, s koeficijentima realnih vrijednosti iz intervala $[-1, 1]$. Funkcija dobrote računata je kao euklidska udaljenost stvarne funkcije i pronađenog rješenja. Na slici 3.17 prikazane su dobivene statistike provedenih eksperimenata.

Problem	Hits %					
	GP		SSHC		SGP	
	avg	sd	avg	sd	avg	sd
Polynomial3	79.9	23.1	100.0	0.0	99.5	1.5
Polynomial4	60.5	27.6	99.9	0.9	99.9	0.9
Polynomial5	40.7	21.6	100.0	0.0	99.5	2.0
Polynomial6	37.5	23.4	100.0	0.0	98.9	3.1
Polynomial7	30.7	18.5	100.0	0.0	99.9	0.9
Polynomial8	34.7	16.0	99.5	2.0	99.7	1.3
Polynomial9	20.7	13.2	100.0	0.0	98.5	4.9
Polynomial10	25.7	16.7	99.4	1.7	99.9	0.9

Slika 3.17: Usporedba učinkovitosti GP-a i GPt-a sa SGP-om i SSHC-om (preuzeto iz [13])

Vidljivo je kako je GP kao i u prethodnom slučaju, gori od SSHC-a i SGP-a, te da je standardna devijacija dosta veća nego kod ostalih.

Programi

Kako bi se pokazala učinkovitost semantičkog križanja nad programskim jedinkama, korišten je problem klasifikacije u jednu klasu na osnovu dvije značajke ($(n_c, n_v) \rightarrow n_{cl}$). Na slici 3.18 prikazani su rezultati eksperimenata. Ovdje je ponovno pokazana inferiornost GP-a naprema SSHC-u i SGP-u.

Problem			Hits %								Length			
			GP		GPt		SSHC		SGP					
n_v	n_c	n_{cl}	avg	sd	avg	sd	avg	sd	avg	sd	GP	GPt	SSHC	SGP
3	3	2	80.00	8.41	97.30	4.78	99.74	0.93	99.89	0.67	1.6	1.9	2.3	2.3
3	3	4	49.15	9.96	78.89	8.93	99.89	0.67	99.00	1.63	1.6	2.1	2.3	2.3
3	3	8	37.04	5.07	59.52	14.26	99.74	0.93	96.04	2.85	1.2	1.9	2.3	2.3
3	4	2	67.92	7.05	93.80	5.41	99.95	0.28	99.58	0.80	1.8	2.3	2.7	2.7
3	4	4	39.11	7.02	68.48	8.66	99.84	0.47	98.08	1.64	1.7	2.3	2.7	2.7
3	4	8	28.02	3.73	46.98	14.48	99.73	0.58	94.22	1.72	1.1	2.0	2.7	2.7
4	3	2	88.31	6.98	98.89	2.89	99.96	0.22	100.00	0.00	1.6	1.9	2.9	2.9
4	3	4	48.85	6.54	88.15	10.10	100.00	0.00	99.54	0.68	1.4	2.2	2.9	2.9
4	3	8	36.54	9.01	60.37	17.14	100.00	0.00	96.63	1.23	1.0	1.9	2.9	2.9
4	4	2	82.75	8.21	99.79	1.12	100.00	0.00	99.86	0.23	2.2	2.3	3.3	3.3
4	4	4	44.13	8.75	77.55	6.30	100.00	0.00	99.68	0.29	2.0	2.4	3.3	3.3
4	4	8	30.63	5.33	50.21	15.08	99.96	0.12	98.84	0.58	1.4	2.1	3.3	3.3

Slika 3.18: Usporedba učinkovitosti GP-a i GPt-a sa SGP-om i SSHC-om(preuzeto iz [13])

Unatoč prikazanim rezultatima, autori [13] tvrde da ovaj operator križanja djeluje

bolje od uobičajenih operatora u genetskom programiranju. No, za to nisu priložili nikakav dokaz. U sklopu ovog rada implementirano je ovakvo križanje, te će u kasnijim poglavljima biti prikazana usporedba učinkovitosti ovog operatora s onim operatorima koji su opisani u prethodnim poglavljima.

Ispitivanja

U ovom poglavlju biti će opisana istraživanja provedena u sklopu ovog rada. Za istraživanja je korišteno radno okruženje ECF (*eng. Evolutionary Computation Framework*) koje je za potrebe ovog rada nadograđeno potrebnim, nedostajućim operatorima križanja.

4.1 ECF

ECF [14] (*eng. Evolutionary Computation Framework*) predstavlja okruženje napisano u jeziku C++ za rješavanje različitih problema evolucijskog računanja. Za potrebe ovog rada, najznačajnije korišten dio ECF-a uključuje stablasti genotip (*eng. tree*), zajedno s njegovim operatorima križanja i mutacije. Dio koji je nedostajao ovom okruženju za provedbu kasnije opisanih eksperimenata uključuje implementaciju homolognog, deterministčkog, probabilističkog i semantičkog križanja. Iz tog razloga, kao praktičan dio ovog rada, ECF je nadograđen sa spomenutim operatorima križanja.

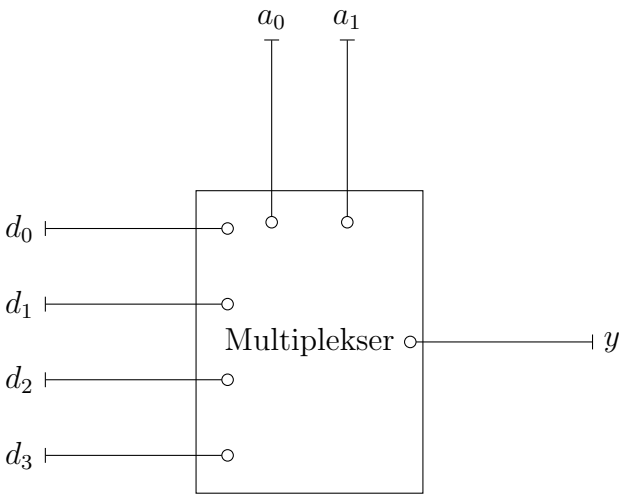
4.2 Problemi za ispitivanje

Kako bi se pokazala učinkovitost različitih operatora križanja nad različitim vrstama problema, ispitivanja su provedena nad različitim problemima simboličke regresije i logičke i programske domene. U nastavku je za svaku vrstu problema opisan specifičan problem koji je kasnije korišten za provedbu istraživanja. Nakon toga dana je detaljna analiza dobivenih rezultata, koja sadrži međusobnu usporedbu učinkovitosti različitih operatora križanja za svaki pojedini problem.

4.2.1 Logičke funkcije

Multiplekser problem

Kako bi se ispitala učinkovitost operatora križanja nad evolucijom logičkih funkcija, provedeni su eksperimenti nad 6 - multiplekser i 11 - multiplekser problemu [2]. Općenito, cilj n - multiplekser problema je pronaći logičku funkciju koja za danih n bitova (od kojih je određen broj adresnih i podatkovnih bitova) generira odgovarajući ulaz. Na slici **??** prikazan je 6 - multiplekser, koji za 2 adresna bita na izlaz propušta jedan od 4 podatkovna bita.



Slika 4.1: 6 - multiplekser

Primjerice, za adresne bitove $a_0 = 1$ i $a_1 = 0$, na izlaz bi se pustio drugi po redu podatkovni bit (01 binarno = 1 decimalno), odnosno d_1.

U tablici 4.1 prikazani su korišteni parametri za predstavljene 6 i 11 - multiplekser probleme.

problem	6 - multiplekser	11 - multiplekser
skup završnih znakova	$\{a_0, a_1, d_0, d_1, d_2, d_3\}$	$\{a_0, a_1, a_2, d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7\}$
skup nezavršnih znakova	$\{AND, OR, NOT, IF\}$	$\{AND, OR, NOT, IF\}$
funkcija dobrote	broj netočnih izlaza	broj netočnih izlaza

Tablica 4.1: Parametri n multiplekser problema

Evolucija kriptografski sigurnih logičkih funkcija

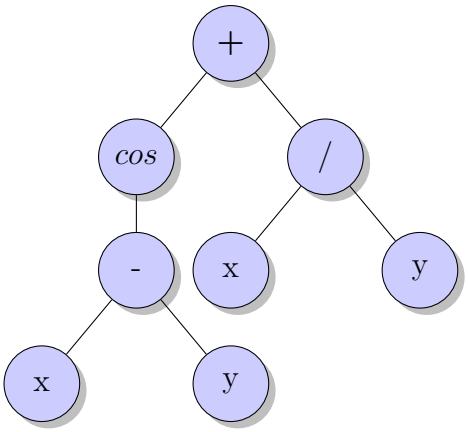
Za ispitivanje učinkovitosti operatora nad logičkim funkcijama također je korišten problem evolucije kriptografski sigurnih logičkih funkcija [15]. Cilj je pronaći neku logičku funkciju koja će imati dobra kriptografska svojstva. Svojstva koja posjeduje dobra kriptografska logička funkcija uključuju:

- balansiranost - funkcija je balansirana ukoliko ima jednak broj istinitih i lažnih vrijednosti
- visoku nelinearnost - linearnost logičke funkcije definira se kao postojanje $a_0, a_1, ..., a_n \in \{0, 1\}$ tako da je $f(b_1, ..., b_n) = a_0 \oplus (a_1 \wedge b_1) \oplus ... \oplus (a_n \wedge b_n)$ za sve $b_1, ..., b_n \in \{0, 1\}$
- visok algebarski stupanj - algebarski stupanj logičke funkcije jednak je broju varijabli izraza s najvećim brojem varijabli u algebarskom normalnom obliku funkcije
- visok algebarski imunitet - algebarski imunitet definiran je kao minimalni broj ne-nul funkcija g takvih da su $fg = 0$ ili $(f \oplus 1)g = 0$
- visok korelacijski imunitet - korelacijski imunitet funkcije f je maksimalna vrijednost m takva da $|\hat{F(\omega)}| = 0$ za sve vrijednosti Hammingove težine $\omega \leq m$

Budući da je nemoguće stvoriti logičku funkciju koja posjeduje sva ova svojstva, algoritmu se postavlja zadatak pronalaska dobre kombinacije nekih od ovih svojstava.

4.2.2 Simbolička regresija

Simbolička regresija je postupak pronalaženja matematičkog izraza iz danih empirijskih podataka. Za potrebe ovih mjerenja korištena je stablasta jedinka koja predstavlja jedan matematički izraz. Taj izraz se iz jedinke može jednostavno isčitati *in-order* obilaskom stabla. Na slici 4.2 prikazana je jedinka koja predstavlja matematički izraz $cos(x - y) + (x/y)$.



Slika 4.2: Primjer jedinke koja rješava problem simboličke regresije koja predstavlja izraz $cos(x - y) + (x/y)$

U tablici 4.2 [1] su dani izrazi koji su bili ciljna funkcija u provedenim istraživanjima, zajedno s njihovim intervalom domene i oznakama koje su kasnije korištene za jednostavnije referenciranje.

oznaka	izraz	interval domene	funkcija dobrote
symb1	$log(x+1) + log(x^2+1)$	$x \in [0, 20]$	mse
symb2	$sin(x) + sin(y^2)$	$x, y \in [-10, 10]$	mse
symb3	$2 \cdot sin(x) \cdot cos(y)$	$x, y \in [-10, 10]$	mse
symb4	$x \cdot y + sin((x+1) \cdot (y-1))$	$x, y \in [-10, 10]$	mse
symb5	$\frac{8}{2+x^2+y^2}$	$x, y \in [-10, 10]$	mse
symb6	$\frac{x^3}{5} + \frac{y^3}{2} - x - y$	$x, y \in [-10, 10]$	mse

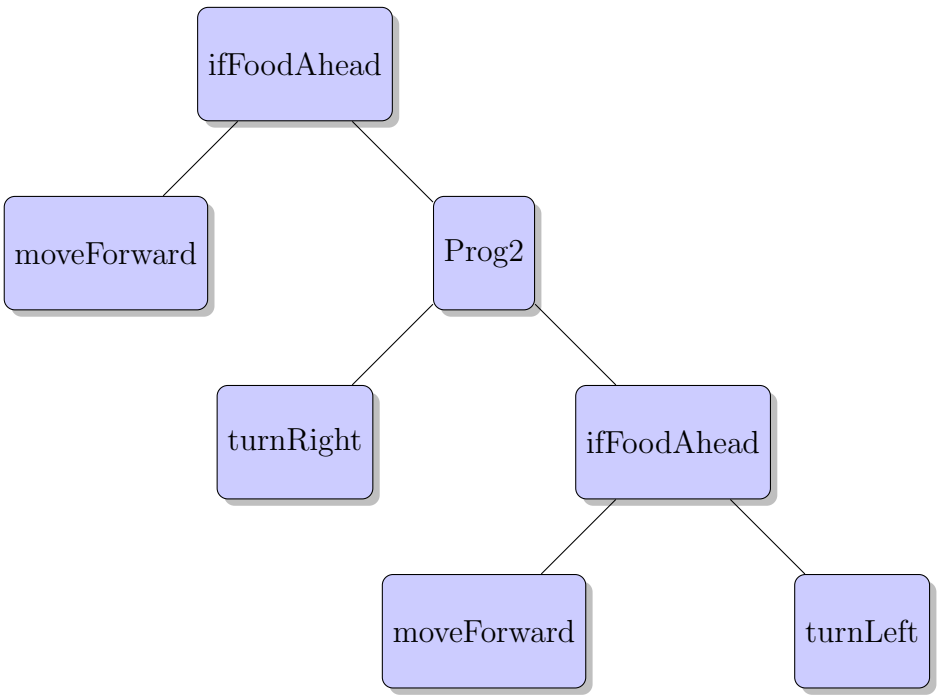
Tablica 4.2: Popis problema simboličke regresije

4.2.3 Programi

Programski problem korišten za evaluaciju operatora križanja bio je problem umjetnog mrava (*eng. Artificial Ant Problem*). Rješenje ovog problema jest evoluirati program koji opisuje ponašanje mrava u nekoj okolini koja sadrži hranu. Dobar program bi trebao biti sposoban pronaći i pojesti što više hrane iz okoline. Jedinka je u ovom slučaju također predstavljena kao stablo. Primjer jedinke prikazan je na slici 4.3.

[1] mse označava kvadratičnu sumu odstupanja (*eng. mean square error*)

Dobrota jedne ovakve jedinke jednaka je broju pojedene hrane u testnoj, prilikom učenja ne viđenoj okolini.



Slika 4.3: Primjer jedinke programa umjetnog mrava

Nezavršni znakovi koji grade jednu ovakvu jedinku su:

1. **If food ahead** - provjerava da li se nalazi hrana ispred mrava - ako da izvršava lijevu granu, ako ne, izvršava desnu granu.
2. **Prog2** - slijedno izvršava lijevu, te zatim desnu granu.
3. **Prog3** - slijedno izvršava (s lijeva na desno) svaku od 3 pridružene grane.

Završni znakovi koji grade ovu jedinku su:

1. **move forward** - pomiče mrava za jedno mjesto unaprijed.
2. **turn right** - okreće mrava u desno.
3. **turn left** - okreće mrava u lijevo.

4.3 Rezultati

Literatura

[1] Ivan Kokan. Primjena genetskog programiranja za rješavanje problema prijanjanja proteina. *Diplomski rad*, 2010.

[2] John R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, 1992.

[3] Marko Čupić; Bojana Dalbelo Bašić; Marin Golub. *Neizrazito, evolucijsko i neuroračunarstvo*. 2013.

[4] S. Luke; L. Spector L. Spector. *A comparison of crossover and mutation in genetic pro- gramming*. 1997.

[5] David R. White; Simon Poulding. *A Rigorous Evaluation of Crossover and Mutation in Genetic Programming*. Dept. of Computer Science, University of York.

[6] http://dces.essex.ac.uk/staff/rpoli/gp-field-guide/24recombinationandmutation.html7_4.

[7] R. Poli; W. B. Langdon W. B. Langdon. A new schema theory for genetic programming with one-point crossover and point mutation. *Genetic Programming 1997: Proceedings of the Second Annual Conference*, 1997.

[8] Riccardo Poli;W.B. Langdon. Genetic programming with one-point crossover and point genetic programming with one-point crossover and point genetic programming with one-point crossover and point genetic programming with one-point crossover and point genetic programming with one-point crossover and point mutation. 1997.

[9] Patrik D'haeseleer. Context preserving crossover in genetic programming. *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, 1994.

[10] W. B. Langdon. Size fair and homologous tree crossovers for tree genetic programming. *Genetic Programming and Evolvable Machines*, 2000.

[11] Riccardo Poli; W.B. Langdon. On the search properties of different crossover operators in genetic programming. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, 1998.

[12] Josh C. Bongard. A probabilistic functional crossover operator for genetic programming. 2010.

[13] Alberto Moraglio; Krzysztof Krawiec; Colin G. Johnson. Geometric semantic genetic programming.

[14] Domagoj Jakobović. Ecf: http://gp.zemris.fer.hr/ecf/.

[15] Stjepan Picek; Domagoj Jakobovic; Marin Golub. Evolving cryptographically sound boolean functions. 2013.